SOLIDIFIED

Audit Report for Dfyn - January 26, 2023

Summary

Audit Report prepared by Solidified covering the Dfyn smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on January 26, 2023, and the results are presented here.

Audited Files

The source code has been supplied in a private source code repository:

https://github.com/dfyn/V2-Contracts

Commit number: `c22ca5c998b5bf4098f8d8c05ceaf909dadeba99`

Update: Fixes were received on Sunday February 19, 2023.
Latest commit number: `17a6628227ac951d5fa974b0718e6da7c6247bbd`

Intended Behavior

Dfyn is a concentrated liquidity pool automated market maker (AMM) that implements fully on-chain limit orders.

Findings

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium	-
Level of Documentation	Medium	-
Test Coverage	High	-

Issues Found

Solidified found that the Dfyn contracts contain no critical issues, 3 major issues, 14 minor issues, and 10 informational notes.

We recommend issues are amended, while informational notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	ConcentratedLiquidityPool.sol: Function initialize() can be called multiple times	Major	Resolved
2	Vault.sol: An incorrect data.balance is being maintained for vault strategy	Major	Resolved
3	ConcentratedLiquidityPool.sol: Function collectLimitOrderFee() breaks reserves invariant leading to reverts in mint() and createLimitOrder()	Major	Resolved
4	MasterDeployer.sol/ConcentratedLiquidityPool.sol: The protocol risks permanently losing access to its collected fees	Minor	Resolved
5	MasterDeployer.sol: Function setDfynFee() emits the incorrect event when limitOrderFee is set	Minor	Resolved
6	Farm.sol: Function subscribe() does not validate the given array lengths	Minor	Resolved
7	LimitOrderManager.sol: Function createLimitOrder() does not refund any extra ETH sent	Minor	Resolved
8	DfynLPToken.sol: Unsafe minting in mint() function	Minor	Resolved
9	MixedRouteQuoterV1.sol: Possible underflow in casting from int256 to uint256	Minor	Resolved
10	ConcentratedLiquidityPoolFactory.sol: Possible	Minor	Resolved

	zero address for liquidity pool tokens		
11	ConcentratedLiquidityPoolHelper.sol: Possible invalid limitTick value	Minor	Resolved
12	DfynRouterV2.sol: Unused amountOut parameter in function swapCallBack()	Minor	Resolved
13	TickCounter.sol: Incorrect while statement in function countTicks()	Minor	Resolved
14	MasterDeployer.sol: limitOrderFee is not validated in constructor	Minor	Resolved
15	LimitOrderManager.sol: Admin can set limitOrderCharge to arbitrarily large numbers	Minor	Resolved
16	SwapExcecuter.sol: Fees are not added for most common case of zero-for-one exact output swaps	Minor	Resolved
17	MasterDeployer.sol: Function setDfynFee() can lower limitOrderFee so it is lower than LimitOrderManager.rebateRate for pool	Minor	Resolved
18	Farm.sol: Constructor does not validate _vault / _limitOrderManager	Note	Resolved
19	ConcentratedLiquidityPool.sol: Incorrect documentation comment for MIN_FEE	Note	Resolved
20	DfynRouter.sol: Function sweep() is unnecessarily declared as payable	Note	Resolved
21	LimitOrderManager.sol: Function setLimitOrderCharge() does not emit an event	Note	Resolved
22	Farm.sol: Function subscribe() can potentially save on gas by declaring positionIds as calldata	Note	Resolved
23	MixedRoutedQuoterV1.sol: Unnecessary use of require statement	Note	Resolved
24	LimitOrderManager.sol: Unnecessary use of return statement	Note	Resolved

25	LimitOrderManager.sol: Use of confusing function name	Note	Resolved
26	The deadline field of various swap structs is unused	Note	-
27	ConcentratedLiquidityPool.sol: Unused storage variables	Note	Resolved

Critical Issues

No critical issues have been found.

Major Issues

1. ConcentratedLiquidityPool.sol: Function initialize() can be called multiple times

The function `initialize()` can be called an unlimited number of times by the owner, thus potentially compromising several of the contract's immutables such as `masterDeployer` and `vault`.

Recommendation

Only allow `initialize()` to be called once in the lifetime of the contract.

Status

Resolved

2. Vault.sol: An incorrect data.balance is being maintained for vault strategy

In the harvest() function, when a strategy makes a profit, the data.balance variable is never updated to reflect this. The data.balance variable is then used later in the function to determine whether to further invest if (data.balance < targetBalance) or divest else if (data.balance > targetBalance) in the strategy. If data.balance is not updated after a profit, users will be misled into further investing into a strategy when they could have divested their profit earnings instead.

Recommendation

Add a statement to update `data.balance` when a strategy has turned a profit.

Status

Resolved

3. ConcentratedLiquidityPool.sol: Function collectLimitOrderFee() breaks reserves invariant leading to reverts in mint() and createLimitOrder()

The invariant `reserve0 + limitOrderReserve0 == _balanceOf(token0)` (similarly for `token1`) must hold before and after the call to `collectLimitOrderFees()`.

However, at the end of the function call, the right hand side of the invariant is now `_balance(token1) - (token1LimitOrderFee - token1Rebate)`.

The left hand side of the invariant is too large, by an amount equal to `token1LimitOrderFee - token1Rebate`.

The impact of the invariant being broken is that further calls to `mint()` and `createLimitOrder()` will fail because they both check the invariant and revert otherwise. The only way to fix this problem is to transfer additional tokens manually to the pool which results in a loss of funds.

Recommendation

The left hand side of the invariant must be decreased by the same amount. The function should be updated as follows:

```
    function collectLimitOrderFee() public lock returns (uint256 amount0, uint256 amount1) {
        if (token0LimitOrderFee > token0Rebate) {
            amount0 = token0LimitOrderFee - token0Rebate;
```

```
            token0LimitOrderFee = token0Rebate;

            reserve0 -= uint128(amount0);

            _transfer(token0, amount0, dfynFeeTo, false);

        }

        if (token1LimitOrderFee > token1Rebate) {

            amount1 = token1LimitOrderFee - token1Rebate;

            token1LimitOrderFee = token1Rebate;

            reserve1 -= uint128(amount1);

            _transfer(token1, amount1, dfynFeeTo, false);

        }

    }
```

Intuitively, it makes sense that the reserves must go down since the limit order fees come from swappers who fulfill the limit orders. The fees must be paid from the regular reserves and not the limit order reserves since the limit order reserves can be emptied on a claim.

Also, note that the condition is now `if (token0LimitOrderFee > token0Rebate)`. This is because rebates need to be paid to those who claim the limit orders. These rebates must stay in the pool.

Note

The same issue also exists in `ConcentratedLiquidityPool.claimLimitOrder()`.

Status

Resolved

Minor Issues

4. MasterDeployer.sol/ConcentratedLiquidityPool.sol: The protocol risks permanently losing access to its collected fees

In case the keys for `dfynFeeTo` are ever lost, there is no way to update its value in the contract, thus rendering the protocol fees permanently inaccessible.

Recommendation
Implement a setter function for `dfynFeeTo`.

Status
Resolved

5. MasterDeployer.sol: Function setDfynFee() emits the incorrect event when limitOrderFee is set

The function `setDfynFee()` incorrectly emits the `DfynFeeUpdated` event when `limitOrderFee` is set.

Recommendation
Emit the `LimitOrderFeeUpdated` event instead.

Status
Resolved

6. Farm.sol: Function subscribe() does not validate the given array lengths

The function `subscribe()` does not validate that `positionIds` and `incentiveIds` are of the same length.

Recommendation
Validate the given array lengths in order to avoid unintended input mistakes.

Note
The same issue exists in functions `Farm.claimRewards()` and `Vault.batchFlashLoan()`.

Status
Resolved

7. LimitOrderManager.sol: Function createLimitOrder() does not refund any extra ETH sent

The function `createLimitOrder()` does not refund any extra ETH sent by mistake, resulting in the funds being stuck in the contract.

Recommendation
Refund any ETH sent that exceeds the value of `amountIn`.

Note
The same issue exists in: `DfynRouter.exactInput()`, `DfynRouter.exactInputSingleWithNativeToken()`, `DfynRouter.exactInputWithNativeToken()`, `DfynRouter.complexPath()`, `DfynRouterV2.exactInputSingle()`, `DfynRouterV2.exactInput()`, `DfynRouterV2.exactOutputSingle()`, `DfynRouterV2.exactOutput()`, and `ConcentratedLiquidityPoolManager.mint()`.

Status

Resolved

8. DfynLPToken.sol: Unsafe minting in mint() function

OpenZeppelin discourages the use of `_mint()` , as it does not check to ensure that the recipient is a smart contract that implements the ERC721Receiver interface. This could result in an NFT being lost in a contract.

Recommendation

Use the _safeMint() function instead of `_mint()`.

Status

Resolved

9. MixedRouteQuoterV1.sol: Possible underflow in casting from int256 to uint256

The uniswapV3SwapCallback() function expects that if amount0Delta is negative when amount1Delta is positive and vice versa. However, a user can enter a positive number for both parameters which will result in an underflow of the uint256 cast on line 127 returning an extremely large value for amountReceived.

Recommendation

Add a check to ensure that if amount0Delta > 0 then amount1Delta < 0.

Status

Resolved

10. ConcentratedLiquidityPoolFactory.sol: Possible zero address for liquidity pool tokens

---

In the deployPool() function there is no check to ensure that tokenA and tokenB are valid addresses. For e.g., if tokenA is address(0) and tokenB is a valid address the check on line 24 will pass. The call to _registerPool() on line 44 will not catch this case also.

Recommendation

Add a check to ensure that both tokenA and tokenB are not the zero address.

Status

Resolved

11. ConcentratedLiquidityPoolHelper.sol: Possible invalid limitTick value

---

In the getLowerOldAndUpperOldLimit() function, the check on line 63 does not ensure that the limitTick value is within the range TickMath.MIN_TICK to TickMath. MAX_TICK. This allows a user to enter a value for limitTick that can be smaller than TickMath.MIN_TICK or larger than TickMath. MAX_TICK.

Recommendation

Change the check to require(limitTick > TickMath.MIN_TICK && limitTick < TickMath.MAX_TICK, "Invalid limit tick").

Status

Resolved

12. DfynRouterV2.sol: Unused amountOut parameter in function swapCallBack()

The swapCallBack() function passes in an amountOut parameter, but the parameter is never used within the function.

Recommendation

Validate the call to exactOutputInternal() against amountOut.

Status

Resolved

13. TickCounter.sol: Incorrect while statement in function countTicks()

In the countTicks() function, if tickBefore is greater than tickAfter, then count will always return zero. This is because the while statement in the else block checks for ticksCrossed <= tickAfter where ticksCrossed is initialized to tickBefore. Hence, the loop will never start.

Recommendation

Change the while statement to read ticksCrossed >= tickAfter.

Status

Resolved

14. MasterDeployer.sol: limitOrderFee is not validated in constructor

---

There is no check to ensure that limitOrderFee <= MAX_FEE. There is also an invalidLimitOrderFee() error on line 11 that is not being used.

Recommendation

Add a check to ensure limitOrderFee <= MAX_FEE and use the associated error message.

Status

Resolved

15. LimitOrderManager.sol: Admin can set limitOrderCharge to arbitrarily large numbers

---

There is no maximum value that parameter `_fee` can be when parameter `_isRebate == false` for function `LimitOrderManager.setLimitOrderCharge()`. This can lead to arbitrarily high charges being forced upon creators of limit orders.

Recommendation

Add a maximum charge constant and check that `_fee` is below it similar to the checks in function `ConcentratedLiquidityPool.updateSwapFee()`.

Status

Resolved

16. SwapExcecuter.sol: Fees are not added for most common case of zero-for-one exact output swaps

---

The function `_swapExactOut()` is missing the addition of fees for the case of zero-for-one exact output swaps that don't cross ticks. The impact is that fees are not collected. As this is a common case this could mean substantial loss of income for the Dfyn protocol. Also, fee-less swapping makes profiting from arbitrage possible for even smaller price differences than is normally possible.

Recommendation

Add the following line between lines `SwapExecuter.sol:66` and `67`

```
cache.amountIn = cache.amountIn + swapExecute.fee;
```

Status

Resolved

17. MasterDeployer.sol: Function setDfynFee() can lower limitOrderFee so it is lower than LimitOrderManager.rebateRate for pool

---

A check in `LimitOrderManager.sol:232` ensures that the rebate rate set for a particular pool (`rebateRate[pool]`) is less than or equal to `pool.limitOrderFee()`. However, it would be possible for a call to `MasterDeployer.setDynFee()` to lower the global `limitOrderFee` and a subsequent call to function `updateProtocolFee()` on the pool to set its `limitOrderFee` to a value lower than the corresponding `rebateRate[pool]` value.

The impact of this issue is that there may be insufficient fees collected to pay for the rebates when limit orders are claimed. Although this is a significant issue it is unlikely to happen in practice hence this issue has been classified as minor.

Recommendation

There is no simple fix for this since the rebate rate is stored in a mapping in the LimitOrderManager and not the pool itself. Instead of a mapping from pool to rebate rate being stored in the `LimitOrderManager`, consider having the `rebateRate` stored in the `ConcentratedLiquidityPool` contract. Then, when updateProtocolFee is called, ensure that one sets `rebateRate = min(rebateRate, limitOrderFee)`.

Status

Resolved

Informational Notes

18. Farm.sol: Constructor does not validate _vault / _poolManager

The `Farm` contract constructor does not check if the returned `_vault` has value or if `_poolManager` has a valid value.

Recommendation

Consider checking that `_vault != address(0)` and `_poolManager != address(0).`

Status

Resolved

19. ConcentratedLiquidityPool.sol: Incorrect documentation comment for MIN_FEE

The documentation comment for `MIN_FEE` seems to belong to `MAX_FEE`.

Recommendation

Consider providing the correct documentation for the minimum fee value.

Status

Resolved

20. DfynRouter.sol: Function sweep() is unnecessarily declared as payable

---

There is no reason where the caller ever needs to send ETH to the `sweep()` function.

Recommendation

Consider removing `payable` from the function's declaration to avoid any unintended behavior.

Status

Resolved

21. LimitOrderManager.sol: Function setLimitOrderCharge() does not emit an event

---

Recommendation

Consider emitting an event in function setLimitOrderCharge().

Status

Resolved

22. Farm.sol: Function subscribe() can potentially save on gas by declaring positionIds as calldata

---

Declaring the parameter `positionIds` as `calldata` instead of `memory` can potentially save on gas, since the array values would be directly read from `calldata` instead of being copied to `memory` first.

Recommendation

Declare the `positionIds` parameter as `calldata` instead of `memory` to save on gas fees.

Status

Resolved

23. MixedRoutedQuoterV1.sol: Unnecessary use of require statement

---

In the swapCallBack() function on line 112 there is a require() statement that is immediately followed by a revert statement.

Recommendation

Consider removing the require() statement since the function will revert anyway.

Status

Resolved

24. LimitOrderManager.sol: Unnecessary use of return statement

In the claimLimitOrder() function on line 165 it says if (amount == 0) return. However, on the previous line there is require (amount > 0). This makes line 165 irrelevant.

Recommendation

Consider removing the if statement from line 165.

Status

Resolved

25. LimitOrderManager.sol: Use of confusing function name

The getLimitOrderTokenBalance() function name indicates that a numerical balance will be returned. Instead, the function actually returns the owner address of a specific token id.

Recommendation

Consider renaming the function to reflect its actual functionality.

Status

Resolved

26. The deadline field of various swap structs is unused

The deadline field of the following structs is unused.

- `ExactInputSingleParams`

- `ExactInputParams`
- `ExactOutputSingleParams`
- `ExactOutputParams`

Recommendation

Consider removing the unused `deadline` field.

27. ConcentratedLiquidityPool.sol: Unused storage variables

Storage variables `rebateRate` and `limitOrderCharge` are currently unused in contract `ConcentratedLiquidityPool`. This is confusing (and error prone) since there are two mappings of exactly the same name in `LimitOrderManager`.

Status

Resolved

Audit Report for Dfyn - January 26, 2023

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Dfyn or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Oak Security GmbH